A Denotational Model of Type Inheritance and Generics

AUTHOR(S):

Tsuiki, Hideki

CITATION:

Tsuiki, Hideki. A Denotational Model of Type Inheritance and Generics. 数理解析研究所講究録 1989, 709: 116-142

ISSUE DATE:

1989-12

URL:

http://hdl.handle.net/2433/101658

数理解析研究所講究録
第709巻 1989年 116-142

116

A Denotational Model of Type Inheritance and Generics

Hideki Tsuiki
Research Institute for Mathematical Sciences
Kyoto University
Kitashirakawa, Sakyo, Kyoto 606, Japan

Abstract

A denotational semantics is given to type inheritance and higher order generic functions, where a type is defined to inherit another type if there exists a coercion function between them and a function is defined to be generic if it preserves types and coercions. The semantic domain is constructed in the category of I-domains (domain with type inheritance), whose objects are mathematical models of domains with hierarchical type structure, and whose morphisms are mathematical models of generic functions. This category is cartesian closed and such domain equations as $M = M_B + [M \to M]$ are solvable in this category. On the solution of this equation, the semantics of a simple functional language with hierarchical type structure and higher order generic functions is defined.

1. Introduction

Coercion and type inheritance

Most programming languages have type structures. Values are classified into types according to their representation on memory and the operations applicable to them. Many programming languages, in addition, have type structures with inheritance.

There seems to be several viewpoints on the meaning of type inheritance. [Car85] constructs a denotational model of type inheritance, in which a type is a non-empty weak ideal of the value domain, and inheritance relation is defined by set inclusion between ideals. The denotational model we construct is essentially different. Types correspond to disjoint sets of values, that is, each value belongs to only one type; and inheritance relation is defined by the existence of a coercion function from the more specified type to the more general type, instead of by set inclusion.

As an example, consider the inheritance relation between `int` and `real` which many programming languages have. In those languages, `1` which belongs to `int` and `1.0` which belongs to `real` are treated as separate values, and when a function defined on `real` is applied to `1`, it is coerced to `1.0` automatically and the function is applied to `1.0`. In this case, `int` can be regarded as inheriting `real` through this coercion function from `int` to `real`.

As another example, consider record types with typed slots [†1]. Record types are important because they are used as classes in some dialects of lisp with object oriented facilities. Consider the following types:

```
deftype person <age  → integer,
                name → string,
                sex  → boole  >

deftype female-student <age    → integer,
                        name   → string,
                        sex    → true,
                        school → string  >
```

[†1] Though we do not deal with record types in order to make the theory simple, the contents of this paper can be easily extended to record types.

where `true` is a type which inherits `boole` and contains only `t`, the value which means female in this example. For each value belonging to `female-student`, there exists a value belonging to `person`, where the coercion from `female-student` to `person` is obtained by ignoring the value attached to the slot `school` and by coercing the value attached to the slot `sex` from `true` to `boole`. Thus `female-student` inherits `person` through this coercion function [†2].

Consider the following additional type definitions:

```
deftype female <age  → integer,
                name → string,
                sex  → true  >

deftype student <age    → integer,
                 name   → string,
                 school → string  >.
```

`Female-student` inherits both `female` and `student`. Thus record types shown here have multiple inheritance, which means that, by defining order relation $\geq$ as $A \geq B$ when the type A inherits the type B, the set of types is a partially ordered set (in short a poset) [†3]. Moreover, `female-student` is the least upper bound of `female` and `student`, that is, any type which inherits both `female` and `student` inherits `female-student`. In this example of record types, a set of types which has an upper bound has the least upper bound. We call posets with this property *semi-coherent*. As for the value domain, the set of values is also a poset with the order relation $\succeq$ defined as $a \succeq b$ when a is coercible to b. It is also semi-coherent, that is, when a set of values has an upper bound, it has the least upper bound. The semi-coherentness ensures that $((a \sqcup b) \sqcup c)$, if exists, is equal to $(a \sqcup (b \sqcup c))$ in value and type domain, where we write $a \sqcup b$ for the least upper bound of a and b.

We construct, as a semantic domain, a triple (D, T, τ) [†4], where D is the set of values, T is the set of types, and τ is the function from D to T which assigns a type to each value. T and D are semi-coherent posets, and τ is a monotonic function which is surjective and satisfies some conditions in order that the coercion of a value to a type is uniquely determined. We also require that D be a complete partial order (in short cpo), which we need in solving domain equations. We call the triple (D, T, τ) which satisfies these conditions an I-domain – domain with type inheritance. We call D the value domain and T the type domain. We say that x belongs to t when $\tau(x) = t$.

Generic functions

A Generic function is a function that is applicable to more than one types and whose behavior depends on the type of the arguments supplied to it. We can find generic functions in many practical languages with type inheritance. For example, in some languages, the addition function + with two arguments is applicable to arguments both in `int` and `real`, and returns a value in `int` when both of the arguments are `int`, and `real` when either of the arguments is `real`. As another example, there is an extension of lisp called CLOS (Common Lisp Object System), which is the standardization of merging object oriented programming to Common Lisp [BDG87]. CLOS implements message passing using generic functions.

In the model (D, T, τ), we are only concerned with the function space which consists of total generic functions from D to D. In order to treat partially defined generic functions as total generic

---

[†2] The coercion function from `female-student` to `person` is not injective, which did not appear in the example of `real` and `int`.

[†3] Note that we define the order relation in such a way that bigger types have more information.

[†4] We use *italic* fonts for semantic objects.

functions, we consider a special type *null* which inherits every type and a special value *nil* belonging to *null*. Thus, a generic function defined only for some types can be extended to a total generic function by assigning *nil* as the return value for illegal arguments.

Generic functions are very useful in writing programs. However, if a generic function behaves completely differently for different types of arguments, it will cause programming errors which are difficult to detect. In fact, it is difficult to give clear semantics to generic functions if their actions for different types are unrelated. Therefore, we restrict generic functions from D to D so that they preserve types and coercions; where we say that a function preserves types if it maps values belonging to the same type to values belonging to the same type, and that a function preserves coercions if it maps coercible values to coercible values. Thus, generic functions are not simply overloaded functions. We also require that generic functions be continuous, which we need in solving domain equations. We call a continuous function from D to D which preserves types and coercions a generic function of I-domains with respect to T and τ .

[Rey81] gives semantics to generic functions using natural transformations, where generic functions are treated not as values but as operators; so they cannot be used as arguments to other generic functions. The I-domain (D, T, τ) we want to construct as a semantic domain is the one in which a generic function from D to D is treated as an element of D and can be used as an argument or a return value of other generic functions. In other words, we consider *higher order generic functions*. In order to do that, we must assign a type to each generic function. In the model of typed lambda calculus, the type of a function is given by a pair of types: the source type and the destination type. The type of a generic function is not a pair of types, but a function from T to T which, to each type A, assigns the type of return values to arguments belonging to A.

We construct, as a semantic domain, a reflective I-domain $M = (D, T, \tau)$ from a given I-domain $M_B = (D_B, T_B, \tau_B)$: T_B consists of basic types such as *int* and *real*, D_B consists of values belonging to T_B such as 1 and 1.0, and τ_B is the type assignment function from D_B to T_B. D consists of D_B and generic functions from D to D, and T consists of T_B and monotonic functions from T to T which are types of generic functions. Therefore, M satisfies the following equations:

$$\begin{array}{ccccc} D & = & D_B & + & [D \to D] \\ \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle B} & & \downarrow{\scriptstyle \tau^*} \\ T & = & T_B & + & [T \to T] \end{array} \qquad (*)$$

where $[D \to D]$ denotes the set of generic functions from D to D with respect to T and τ , $[T \to T]$ denotes the set of monotonic functions from T to T which are types of generic functions, and τ^* is the type assignment function from $[D \to D]$ to $[T \to T]$†.

To give some examples of generic functions, consider an I-domain (D_B, T_B, τ_B); T_B is equal to $\{int, real, null\}$, D_B is equal to $INT + REAL + \{nil\}$ where INT is the set of integers and $REAL$ is the set of real numbers. $\tau_B : D_B \to T_B$ is defined obviously. There is an embedding function from INT to $REAL$, and the collapsing function from $REAL$ to $\{nil\}$. Taking these functions as the coercion functions, the order on T_B is defined as

$$int \geq real \geq null.$$

Suppose that an I-domain $M = (D, T, \tau)$ which satisfies $(*)$ has been constructed.

† In this paper, we also deal with the value $DWrong$ and the type $TWrong$ which mean type mismatch, which is omited here for simplicity.

Let *square* be a function from D to D which calculates the square of the argument and returns it as an integer if the argument belongs to *int*, and as a real number if the argument belongs to *real*, and returns *nil* if the argument does not belong to *int* nor *real*. Clearly, *square* is a generic function. The type of *square* is a function from T to T which returns *int* to *int*, *real* to *real*, and *null* to other arguments. We denote this function by

$$\lfloor int\Uparrow \mapsto int\rfloor \sqcup \lfloor real\Uparrow \mapsto real\rfloor.$$

The meaning of this notation is defined as follows. $\lfloor t\Uparrow \mapsto s\rfloor$ denotes the function which returns s if the argument is bigger than t by the order on T, and returns *null* if it is not.

$$\lfloor t_1\Uparrow \mapsto s_1\rfloor \sqcup \ldots \sqcup \lfloor t_n\Uparrow \mapsto s_n\rfloor$$

denotes the least upper bound of $\lfloor t_1\Uparrow \mapsto s_1\rfloor, \ldots, \lfloor t_n\Uparrow \mapsto s_n\rfloor$ by the order on $[T \to T]$ induced by that on T. We use this notation only when the least upper bound exists.

Instead of adding product types, we treat a generic function with more than one arguments as that with one argument by currying it. Therefore, the generic function + which calculates sum has the type

$$\lfloor int\Uparrow \mapsto \lfloor int\Uparrow \mapsto int\rfloor\rfloor \sqcup \lfloor real\Uparrow \mapsto \lfloor real\Uparrow \mapsto real\rfloor\rfloor,$$

which returns to *int* the function

$$\lfloor int\Uparrow \mapsto int\rfloor \sqcup \lfloor real\Uparrow \mapsto real\rfloor,$$

and returns $\lfloor real\Uparrow \mapsto real\rfloor$ to *real*.

A function which is defined as a lambda expression using generic functions are also generic. For example, let *sqrt* be the generic square root function which belongs to $\lfloor real\Uparrow \mapsto real\rfloor$, and let *trunc* be the generic truncation function which belongs to $\lfloor real\Uparrow \mapsto int\rfloor$. Then

$$lambda(x).((+(trunc(sqrt\ x)))x)$$

is a generic function whose type is

$$\lfloor int\Uparrow \mapsto int\rfloor \sqcup \lfloor real\Uparrow \mapsto real\rfloor.$$

Note that not all the types of generic functions can be written using this notation. For example, the identity function on D defined as $(lambda(x).x)$ is generic and its type is the identity function on T.

Outline of the paper

In order to construct an I-domain which satisfies $(*)$, we first show that I-domains make a category. In this category, a morphism is a pair of functions: one is between value domains which is a generic function, and the other is between type domains which is the type of the generic function. Next, we prove that the triple $[M \to M] = ([D \to D], [T \to T], \tau^*)$ is also an I-domain, and that I-domains make a cartesian closed category with exponent $[\to]$. After that, we solve the domain equation $M = M_B + [M \to M]$, which is equivalent to $(*)$, in this category.

On the I-domain M which is the solution of this equation, we define the semantics of a simple functional language LTI (Language with Type Inheritance) in the denotational manner. The syntax of LTI consists of a triple (Exp, TExp, typeof), which corresponds to (D, T, τ). Exp is the set of expressions which are lambda expressions with constants. TExp is the set of type expressions which

are extension of lambda expressions in that it has constants and that one can express such function on types as

$$\lfloor int \Uparrow \mapsto int \rfloor \sqcup \lfloor real \Uparrow \mapsto real \rfloor.$$

Let Env be the set of environments and let TEnv be the set of type environments. The semantics of LTI is defined by the pair of semantic functions $(\mathcal{E}, \mathcal{E}^T)$. $\mathcal{E}$: Exp $\rightarrow$ (Env $\rightarrow D$) is the semantic function for expressions and $\mathcal{E}^T$: TExp $\rightarrow$ (TEnv $\rightarrow T$) is the semantic function for type expressions. $\mathcal{E}$ and $\mathcal{E}^T$ make the following diagram commute.

$$\begin{array}{ccccc} \mathsf{Exp} & \times & \mathsf{Env} & \xrightarrow{\mathcal{E}} & \mathrm{D} \\ \downarrow \mathsf{typeof} & & \downarrow \# & & \downarrow \tau \\ \mathsf{TExp} & \times & \mathsf{TEnv} & \xrightarrow{\mathcal{E}^T} & \mathrm{T} \end{array}$$

2. I-domain

In this section, we define I-domain (domain with type inheritance) and **DTI** (the category of I-domains), and examine some properties of them.

2.1 I-domain

An I-domain is a mathematical model of values and types with type inheritance, where a type is defined to inherit another type if there exists a coercion function from the set of values belonging to the former type to that of the latter type. Before giving precise definition of an I-domain, let us give another definition of it slightly informally in order to show that it is a natural model of type structure with inheritance.

An I-domain consists of a set D called the value domain, a set T called the type domain, and a surjective function τ from D to T called the type assignment function. Each element of D is called a value and each element of T is called a type. When a value x and a type t satisfy $\tau(x) = t$, we say that x belongs to t. The set of values belonging to a type t is denoted by $Val(t)$. $Val(t)$ is equal to $\tau^{-1}(t)$. $Val(t)$ and $Val(u)$ does not intersect when $t \neq u$.

The type domain has inheritance relations on it. Let us denote $t \geq u$ when t inherits u. We do not deal with a type structure in which two types can inherit each other. Therefore, we require that $\geq$ be a partial order, that is,

$t \geq t$
if $t \geq u$ and $u \geq v$ then $t \geq v$
if $t \geq u$ and $u \geq t$ then $t = u$

for $t, u, v \in T$. We also require that T be semi-coherent.

DEFINITION. A poset O is *semi-coherent* if every subset which has an upper bound in O has the least upper bound in O.

By taking the empty set, semi-coherent poset has the least element. In a poset O, the greatest lower bound of a subset U, if it exists, is equal to the least upper bound of $\{t | t \leq u,\ \forall u \in U\}$. For every non-empty subset U of O, $\{t | t \leq u,\ \forall u \in U\}$ has every $u \in U$ as an upper bound. So when O is semi-coherent, every non-empty subset U of O have the greatest lower bound.

As for values, we require that when a type t inherits another type u, there be a *coercion function* $\delta_{t,u}$ from $Val(t)$ to $Val(u)$ such that

$$\begin{array}{ll} \delta_{t,t} = id_{Val(t)} & \text{(the identity function of } Val(t)) \\ \text{if } t \geq u \text{ and } u \geq v \text{ then } \delta_{u,v} \cdot \delta_{t,u} = \delta_{t,v}. & (*) \end{array}$$

Then we can define an order relation $\succeq$ on D as follows †.

Let t and u be $\tau(x)$ and $\tau(y)$. If $t \geq u$ and $\delta_{t,u}(x) = y$ then $x \succeq y$.

It is easy to verify that $\succeq$ is an order relation. We require that $(D, \succeq)$ be semi-coherent. We also require that D be a cpo, which we need in constructing a reflective I-domain.

One can verify easily that the triple (D, T, τ) satisfying these conditions has the following property:

If $\tau(x) = t$ and $t \geq u$, then there is a unique y such that $\tau(y) = u$ and $x \succeq y$. (**)

This means that the coercion of a value x to a type that the type of x inherits exists and is unique.

So far, we defined the order relation on D using the coercion functions $\delta_{t,u}$. Conversely, suppose that D is a semi-coherent cpo, T is a semi-coherent poset, and τ is a monotonic function from D to T which is surjective and satisfies (**). Then we can define $\delta_{t,u}$ which satisfies (*) by defining $\delta_{t,u}(x)$ as the y in (**), which is uniquely determined by x. So we define I-domains as follows.

DEFINITION. Let $(D, \succeq)$ be a semi-coherent cpo, $(T, \geq)$ be a semi-coherent poset, and τ be a monotonic function from D to T which is surjective and satisfies (**). We call the triple (D, T, τ) an *I-domain* (*domain with type inheritance*). We write y in (**) as $\delta_{t,u}(x)$ and call it the *coercion of x to u.*

$\delta_{t,u}(x)$ is also denoted by $x|_u$. We denote the least upper bound (l.u.b.) of U by $\sqcup U$, and greatest lower bound by $\sqcap U$. We abbreviate $\sqcup\{e|u \in U\}$ to $\sqcup_{u \in U} e$ where e is some expression which depends on u. We also abbreviate it to $\sqcup_u e$ when the set U is apparent.

I-domains have the following properties.

PROPOSITION 1. *Let (D, T, τ) be an I-domain.*
1) Let nil and null be least elements of D and T, then $\tau(nil) = null$. nil is the only value which belongs to null.
2) If $\tau(x) = \tau(y) = t$ and $x \succeq y$, then $x = y$.
3) Let v be the l.u.b. of a subset U of T, and let z be an element belonging to v. Then $z = \sqcup_{u \in U} z|_u$.
4) τ preserves all the existing least upper bounds. That is, when X is a subset of D which has the least upper bound z, $\sqcup \tau(X)$ exists and is equal to $\tau(z)$.
5) τ preserves greatist lower bounds. That is, when X is a subset of D, $\tau(\sqcap X) = \sqcap \tau(X)$.

(*proof*) 1) It is obvious because τ is surjective and monotonic.

2) Clear from the uniqueness of $\delta_{t,t}(x)$.

3) Since $z \succeq z|_u$ $(\forall u \in U)$ and D is semi-coherent, there exists $x = \sqcup_u z|_u \preceq z$. Since τ is monotone, $\tau(x)$ satisfies $v \geq \tau(x)$ and $\tau(x) \geq u$ $(\forall u \in U)$. So $\tau(z) = v = \tau(x)$, which means $z = x$ by 2).

4) Since $\tau(z) \geq \tau(x)$ $(\forall x \in X)$, $v = \sqcup \tau(X)$ exists and $\tau(z) \geq v$. We have $z \succeq z|_v \succeq z|_{\tau(x)} = x$ $(\forall x \in X)$, which means $z = z|_v$ and so $\tau(z) = v$.

5) We must show that $\tau(\sqcup\{z|z \preceq x,\ \forall x \in X\}) = \sqcup\{u|u \leq \tau(x),\ \forall x \in X)\}$, which is obvious using 4).

(*end of proof*)

† We use $\geq$ for type domain, and $\succeq$ for value domain.

3) means that if v is the l.u.b. of a set of types U, then every value belonging to v is uniquely determined by its coercions to types in U. 4) means that the type of the l.u.b. is determined by the types of each value.

2.2 Generic functions

DEFINITION. Let (D,T,τ), (D',T',τ') be I-domains. A function f from D to D' is *generic with respect to* T,τ,T',τ' if f is continuous and preserves types. Here we say that f preserves types if $\tau(x)=\tau(y)$ implies $\tau'(f(x))=\tau'(f(y))$ for $x,y\in D$.

A generic function is monotonic because it is continuous, which means that if $\tau(x)\geq u$ then $f(x|_u) = f(x)|_{\tau'(f(x|_u))}$. In other words, f preserves coercion. Preserving types and coercions are natural properties of generic functions as we mentioned in Introduction. Generic functions are defined to be continuous, which is a stronger condition than being monotonic. We need this condition in constructing a reflective I-domain.

Let $M=(D,T,\tau)$, $M'=(D',T',\tau')$ be I-domains, and let f be a generic function from D to D' with respect to T,τ,T' and τ'. Then we can define a function $\tau\tau'^*(f)$ from T to T' as follows.

$$\tau\tau'^*(f)(t)=\tau'(f(x)) \qquad \text{where} \quad \tau(x)=t.$$

Here, we can take x which belongs to t because τ is surjective, and $\tau\tau'^*(f)(t)$ does not depend on the choice of x because f preserves types. We call the function $\tau\tau'^*(f)$ the type of f. We denote $\tau\tau'^*(f)$ by $\tau^*(f)$ when M is equal to M'. $\tau\tau'^*(f)$ makes the following diagram commute.

$$\begin{array}{ccc} D & \xrightarrow{\ f\ } & D' \\ \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau'} \\ T & \xrightarrow{\ \tau\tau'^*(f)\ } & T' \end{array}$$

Conversely, $\tau\tau'^*(f)$ is the only function from T to T' that makes this diagram commute, which leads to the following definition of a homomorphism between I-domains.

DEFINITION. Let $M=(D,T,\tau)$, $M'=(D',T',\tau')$ be I-domains. We call a pair of functions (ϕ^D,ϕ^T) a *homomorphism from M to M'* if ϕ^D is a continuous function from D to D' and ϕ^T is a function from T to T' which satisfy $\phi^T\cdot\tau=\tau'\cdot\phi^D$.

When f is a generic function, $(f,\tau\tau'^*(f))$ is a homomorphism. Conversely, it is easy to check that when (ϕ^D,ϕ^T) is a homomorphism, ϕ^D is a generic function and ϕ^T is the type of ϕ^D. So there is a one-to-one correspondence between homomorphisms and generic functions. Homomorphisms satisfy the following Proposition.

PROPOSITION 2. *Let (ϕ^D,ϕ^T) be a homomorphism from $M=(D,T,\tau)$ to $M'=(D',T',\tau')$.*
1) ϕ^T preserves limits of directed sets. That is, if U is a directed subset of T which has $\sqcup U$, then $\sqcup\phi^T(U)$ exists and is equal to $\phi^T(\sqcup U)$. Particularly, it is monotonic.
2) Suppose that $x\in D$, $u\in T$, $\tau(x)\geq u$, then $\phi^D(x)|_{\phi^T(u)}=\phi^D(x|_u)$.

(*proof*) 1) Because τ is surjective, there exists a value c which belongs to $\sqcup U$. We have $\phi^T(\sqcup U)=\phi^T(\tau(c))=\tau'(\phi^D(c))$. On the other hand, $c=\sqcup_u(c|_u)$ by Proposition 1 (3) and so

$$\tau'(\phi^D(c))=\tau'(\phi^D(\sqcup_u(c|_u)))$$

Since $\{c|_u \mid u \in U\}$ is directed and ϕ^D is continuous,

$= \tau'(\sqcup_u (\phi^D(c|_u)))$

$= \sqcup_u (\tau'(\phi^D(c|_u)))$; Proposition1(4)

$= \sqcup_u (\phi^T(\tau(c|_u)))$

$= \sqcup_u (\phi^T(u))$.

2) Since ϕ^D is a generic function, $\phi^D(x|_u) = \phi^D(x)|_{\tau'(\phi^D(x|_u))}$. $\tau'(\phi^D(x|_u))$ is equal to $\phi^T(u)$ from the definition of homomorphisms.

(end of proof)

PROPOSITION 3. *I-domains with homomorphisms make a category with the composition defined as*

$$(\phi^D, \phi^T) \cdot (\phi^{D'}, \phi^{T'}) = (\phi^D \cdot \phi^{D'}, \phi^T \cdot \phi^{T'})$$

and the identity homomorphism of $M = (D, T, \tau)$ *defined as* $id_M = (id_D, id_T)$ *where* id_D *and* id_T *are identity functions of* D *and* T*, respectively. We call this category* **DTI** *(the category of domains with type inheritance).*

DEFINITION. A homomorphism (ϕ^D, ϕ^T) from $M = (D, T, \tau)$ to $M' = (D', T', \tau')$ is *strict* when ϕ^D maps the bottom of D to the bottom of D'.

When (ϕ^D, ϕ^T) is strict, ϕ^T maps the bottom of T to the bottom of T'.

DEFINITION. A homomorphism (ϕ^D, ϕ^T) from $M = (D, T, \tau)$ to $M' = (D', T', \tau')$ is *additive* when ϕ^D preserves all the existing l.u.b's. That is, if X is a subset of D for which $\sqcup X$ exists then $\sqcup \phi^D(X)$ exists and is equal to $\phi^D(\sqcup X)$.

By taking the empty set as X, additive homomorphisms are strict. When (ϕ^D, ϕ^T) is additive, ϕ^T also preserves all the existing l.u.b's.

2.3 Some properties of DTI

In this subsection, we prove that **DTI** is a cartesian closed category. First, we discuss products and sums of **DTI**.

Let $M = (D, T, \tau)$ and $M' = (D', T', \tau')$ be I-domains. We define the product of M and M' as $M \times M' = (D \times D', T \times T', \tau \times \tau')$. Here, $D \times D'$ is the product of two cpos, $T \times T'$ is the product of two posets, and $\tau \times \tau'$ is the product of functions between posets. It is easily proved that $M \times M'$ is an I-domain. The projection from $M \times M'$ to M is the pair of projections from $D \times D'$ to D and from $T \times T'$ to T. The projection to M' is defined in the same way. Actually, $M \times M'$ is the categorical product of **DTI**.

We also define the sum of M and M' as $M + M' = (D + D', T + T', \tau + \tau')$. Here, $D + D'$ is the sum of two cpos with their least elements identified, $T + T'$ is the sum of two posets with their least elements identified, and $\tau + \tau'$ is the sum of two strict functions between posets. It is also easy to prove that $M + M'$ is an I-domain. The injection from M to $M + M'$ is defined as the pair of injections from D to $D + D'$ and from T to $T + T'$. The injection from M' is defined in the same way. Actually, $M + M'$ is the categorical sum in $\mathbf{DTI}_\perp$, the subcategory of **DTI** whose morphisms are strict homomorphisms.

Let $\perp_{\mathbf{POSET}}$ be the poset which consists of only one element, and let $\perp_\tau$ be the only function from $\perp_{\mathbf{POSET}}$ to $\perp_{\mathbf{POSET}}$. Then the I-domain $(\perp_{\mathbf{POSET}}, \perp_{\mathbf{POSET}}, \perp_\tau)$ is the terminal object of **DTI**. This I-domain is denoted by $\perp_{\mathbf{DTI}}$.

Next, we prove that the set of homomorphisms make an I-domain. Let $M = (D, T, \tau)$, $M' = (D', T', \tau')$ be I-domains. Let $[D \to D']$ denote the set of generic functions from D to D' with respect to T, τ, T' and τ'. We define order relation $\succeq$ on $[D \to D']$ which is induced by that on D', that is,

$$f \succeq g \Longleftrightarrow f(x) \succeq g(x) \quad (\forall x \in D).$$

$\succeq$ is an order relation.

PROPOSITION 4. *1)* $[D \to D']$ *is a cpo.*
2) $[D \to D']$ *is semi-coherent.*

(*proof*) 1) Let F be a directed subset of $[D \to D']$. We define $\sqcup F$ as $\sqcup F(x) = \sqcup_{f \in F}(f(x))$ (since F is directed, $\{f(x) | f \in F\}$ is directed for every x). It is easy to prove that $\sqcup F$ is continuous and preserves types. It is clear that $\sqcup F$ is the l.u.b. of F.

2) Let F be a subset of $[D \to D']$ which has an upper bound g. $g(x) \succeq f(x)$ $(\forall f \in F)$. So there exists the least upper bound $c(x)$ of $\{f(x) | f \in F\}$ for each x. It is easy to prove that c is continuous and preserves types. It is clear that c is the l.u.b. of F.

(*end of proof*)

Let $\langle T \to T' \rangle$ be the poset which consists of monotonic functions from T to T'. The order relation on $\langle T \to T' \rangle$ is induced by that on T'. There exists a function $\tau\tau'^*$ from $[D \to D']$ to $\langle T \to T' \rangle$. Though $\tau\tau'^*$ is monotonic, the triple $([D \to D'], \langle T \to T' \rangle, \tau\tau'^*)$ is not an I-domain because $\tau\tau'^*$ is not surjective. An example that it is not surjective is given in Appendix. We need to restrict the type domain to the image of $\tau\tau'^*$. Let $[T \to T']$ be the image of $\tau\tau'^*$. We prove that $[T \to T']$ is semi-coherent.

LEMMA. *Let* $f \in [D \to D']$ *and* $u \in [T \to T']$ *such that* $\tau\tau'^*(f) \succeq u$. *Then* $f|_u : D \to D'$ *defined as* $f|_u(x) = f(x)|_{u(\tau(x))}$ *is a generic function.*

(*proof*) First, we prove that $f|_u$ is continuous. Let X be a directed subset of D.

$$\begin{aligned} f|_u(\sqcup X) &= f(\sqcup X)|_{u(\tau(\sqcup X))} \\ &= (\sqcup f(X))|_{u(\tau(\sqcup X))}. \end{aligned}$$

Since u is in the image of $\tau\tau'^*$, $u(\tau(\sqcup X)) = u(\sqcup \tau(X)) = \sqcup u(\tau(X))$ by Proposition 2(1). Since $\sqcup f(X) \succeq \sqcup f|_u(X)$, we only need to show that $\tau'(\sqcup(f|_u(X))) = \sqcup u(\tau(X))$, which is trivial because $\tau'(\sqcup(f|_u(X))) = \sqcup \tau'(f|_u(X)) = \sqcup u(\tau(X))$. $f|_u$ preserves types because $\tau'(f|_u(x)) = u(\tau(x))$, which only depends on $\tau(x)$.

(*end of proof*)

PROPOSITION 5. $[T \to T']$ *is semi-coherent.*

(*proof*) Let U be a subset of $[T \to T]$ and v be an upper bound of U. We have to show that U has the least upper bound in $[T \to T]$. Take $f \in [D \to D']$ which satisfies $\tau\tau'^*(f) = v$. Since $f|_u(x) \preceq f(x) (\forall u \in U)$, there exists $\sqcup_{u \in U}(f|_u(x))$. Define $g(x) = \sqcup_u(f|_u(x))$. We will show that g is a generic function whose type is $\sqcup U$. g is continuous because $g(\sqcup X) = \sqcup_u(f|_u(\sqcup X)) = \sqcup_u \sqcup_x (f|_u(x)) = \sqcup_x g(x)$ for a directed subset X of D. g preserves types because

$$\begin{aligned} \tau'(g(x)) &= \tau'(\sqcup_u(f|_u(x))) \\ &= \sqcup_u \tau'(f|_u(x)) \\ &= \sqcup_u u(\tau(x)), \end{aligned}$$

which only depends on $\tau(x)$. $\tau\tau'^*(g)$ is the least upper bound of U because

$$\begin{aligned}\tau\tau'^*(g)(t) &= \tau'(g(x)) \qquad\qquad (\text{where } \tau(x)=t)\\ &= \sqcup_u\, u(\tau(x))\\ &= \sqcup_u\, u(t)\end{aligned}$$

(*end of proof*)

$[T \to T']$ is not a cpo even if T and T' are. An example that it is not is shown in Appendix.

THEOREM 1. *Let $M = (D,T,\tau), M' = (D',T',\tau')$ be I-domains, then $[M \to M'] = ([D \to D'],[T \to T'],\tau\tau'^*)$ is an I-domain.*

(*proof*) It is sufficient to prove that $\tau\tau'^*$ satisfies $(**)$, which is obvious by the lemma above.

(*end of proof*)

Moreover, **DTI** is a cartesian closed category (in short ccc).

THEOREM 2. **DTI** *is a ccc.*

(*proof*) Let $M = (D,T,\tau), M' = (D',T',\tau'), M_1 = (D_1,T_1,\tau_1), M_2 = (D_2,T_2,\tau_2), M_3 = (D_3,T_3,\tau_3)$ be I-domains. We have proved that there exist products and terminal object. Let $ev^D : [D \to D'] \times D \to D'$ be $ev^D(f,d) = f(d)$, and let $ev^T : [T \to T'] \times T \to T'$ be $ev^T(u,t) = u(t)$. Then ev^D is continuous and $ev^T \cdot \tau\tau'^* \times \tau = \tau' \cdot ev^D$; so $ev = (ev^D, ev^T)$ is a homomorphism from $[M \to M'] \times M$ to M'. Let $f = (f^D, f^T)$ be a homomorphism from $M_1 \times M_2$ to M_3. f^D is from $D_1 \times D_2$ to D_3 and f^T is from $T_1 \times T_2$ to T_3. It is easily proved that Λf^D: the curry of f^D in the category of cpos , is a function to $[D_2 \to D_3]$, and Λf^T: the curry of f^T in the category of posets, is a function to $[T_2 \to T_3]$. It is easy to verify other conditions for $\Lambda f = (\Lambda f^D, \Lambda f^T)$ to be a homomorphism from M_1 to $[M_2 \to M_3]$. $f = ev \cdot (\Lambda f \times id_{M_2})$ because, as mentioned below, it suffices to prove the equality of D-components of the homomorphisms.

(*end of proof*)

There is a faithful functor F from **DTI** to **CPO** such that

$$\begin{aligned}F(M) &= D\\ F(f) &= f^D \quad \text{where } f = (f^D, f^T) \in Hom(M_1, M_2).\end{aligned}$$

This means that when we prove the equality of two homomorphisms, it suffice to prove the equality of D-components. F preserves the terminal object, product and ev. Since **CPO** has a faithful functor to **Set**, **DTI** has a faithful functor to **Set**, that is, **DTI** is a concrete category.

2.4 Limits in DTI

Next, we will see that every ω^{op}-chain $M_B \xleftarrow{f_0} M_1 \xleftarrow{f_1} M_2 \xleftarrow{f_2} \cdots$ has a limit.

PROPOSITION 6. *Let $\Delta = (M_i, f_i)$ $(M_i = (D_i,T_i,\tau_i), f_i = (f_{D_i}, f_{T_i}))$ be an ω^{op}-chain. Then Δ has the limit (M, g_i).*

(*proof*) $(D_i, f_i^D), (T_i, f_i^T)$ are ω^{op}-chains in the category of cpo and poset respectively. Therefore, there exist their limits (D, g_i^D) and (T, g_i^T). D and T are actually constructed as follows,

$$\begin{aligned}D &= \left\{< d_0, d_1, \ldots > \,|d_i \in D_i,\ f_n^D(d_{n+1}) = d_n\ (n = 0,1,2,\ldots)\right\}\\ T &= \left\{< t_0, t_1, \ldots > \,|t_i \in T_i,\ f_n^T(t_{n+1}) = t_n\ (n = 0,1,2,\ldots)\right\}.\end{aligned}$$

Since τ_i are functions between ω^{op}-chains, there exists their limit τ from D to T. τ is actually defined as follows.

$$\tau(< d_0, d_1, \ldots >) =< \tau_0(d_0), \tau_1(d_1), \ldots > .$$

Since τ is not, in general, surjective, we define $\bar{T}$ as the image of τ .

We show that $\bar{T}$ is semi-coherent. Let $U = \left\{ u^{(l)} =< u_0^{(l)}, u_1^{(l)}, \ldots > | l \in L \right\}$ be a subset of $\bar{T}$ which has an upper bound $s = < s_0, s_1, \ldots > \in \bar{T}$. We have to show that U has the l.u.b. in $\bar{T}$. Take $xs =< xs_0, xs_1, \ldots > \in D$ such that $\tau(xs) = s$. By Proposition 2 (2), $f_n^D(xs_{n+1}|_{u_{n+1}^{(l)}}) = xs_n|_{u_n^{(l)}}$. So, $xu^{(l)} =< xs_0|_{u_0^{(l)}}, xs_1|_{u_1^{(l)}}, \ldots >$ is in D. There exists $a_n = \sqcup_l (xs_n|_{u_n^{(l)}})$ because each D_n is semi-coherent.

Define homomorphisms $f_{i,n} = (f_{i,n}^D, f_{i,n}^T) = f_n \cdot f_{n+1} \cdot \ldots \cdot f_{i-1} : M_i \to M_n$ $(n < i)$. $y_n = \sqcup_{i \geq n} f_{i,n}^D(a_i)$ exists because $\left\{ f_{i,n}^D(a_i) \right\}$ is a directed set in D_n. $f_n^D(y_{n+1}) = y_n$ $(n = 0, 1, \ldots)$ because f_n^D is continuous, which means that $y =< y_0, y_1, \ldots >$ is in D . It is easy to verify that y is the l.u.b. of $\left\{ xu^{(l)} \right\}$.

$$\begin{aligned} \tau_n(y_n) &= \tau_n(\sqcup_{i \geq n} f_{i,n}^D(a_i)) \\ &= \sqcup_{i \geq n} \tau_n(f_{i,n}^D(a_i)) \\ &= \sqcup_{i \geq n} f_{i,n}^T(\tau_i(a_i)) \qquad ; f_{i,n} \text{ is a homomorphism.} \\ &= \sqcup_{i \geq n} f_{i,n}^T(\sqcup_l u_i^{(l)}) \end{aligned}$$

it follows that $\tau(y) =< \tau_0(y_0), \tau_1(y_1), \ldots >$ is the l.u.b. of U. From this proof, it is obvious that D is semi-coherent. It is also easy to verify that τ satisfies the condition for I-domains. Now we have proved that $M = (D, \bar{T}, \tau)$ is an I-domain.

$g_i = (g_i^D, g_i^T)$ is a homomorphism from M to M_i and (M, g_i) is the limit of Δ.

(*end of proof*)

The functor F from **DTI** to **CPO** also preserves limits of ω^{op}-chains.

3. Construction of a reflective I-domain

In this section, we construct an I-domain M which satisfies

$$M = M_B + [M \to M]$$

where $M_B = (D_B, T_B, \tau_B)$ is a given I-domain. [SmP82] gives general conditions under which such recursive domain equations are solvable. We construct an I-domain M using their results. Here, we refer to their results as "Facts" and omit proofs. See [SmP82] for their proofs.

DEFINITION. (Definition 5 of [SmP82]) A category, **K**, is an **O**-*category* if and only if (i) every hom-set is a poset in which every ascending ω-sequence has a l.u.b. and (ii)composition of morphisms is an ω-continuous operation with respect to this partial order.

PROPOSITION 7. **DTI** *is an* **O**-*category.*

(*proof*) 1) Let $M_1 = (D_1, T_1, \tau_1)$, $M_2 = (D_2, T_2, \tau_2)$ be I-domains. Hom-set from M_1 to M_2 is isomorphic to $[D_1 \to D_2]$, which is a cpo. Composition of morphisms is ω-continuous because of the definition of the order on $[D_1 \to D_2]$.

(*end of proof*)

DEFINITION. (Definition 6 of [SmP82]) Let $\mathbf{K}$ be an $\mathbf{O}$-category and let $f : A \to B, g : B \to A$ be morphisms such that $g \cdot f = id_A$ and $f \cdot g \preceq id_B$. Then we call (f, g) a *projection pair from A to B*, f an *embedding* and g a *projection.*

It can be proved that one half of a projection pair determines the other. When f is an embedding, we write f^P for the corresponding projection. We write $\mathbf{K}^E$ for the subcategory of $\mathbf{K}$ which has the same objects as $\mathbf{K}$ and which has only embeddings as morphisms.

FACT 1. (Theorem 1 of [SmP82]) *Let $\mathbf{K}$ be an $\mathbf{O}$-category which has a terminal object, $\perp$, and in which every hom-set $hom(A, B)$ has a least element, $\perp_{A,B}$. Suppose too that composition is left-strict in the sense that for any $f : A \to B$ we have $\perp_{B,C} \cdot f = \perp_{A,C}$. Then $\perp$ is the initial object of $\mathbf{K}^E$.*

PROPOSITION 8. *$\mathbf{DTI}^E$ has an initial object.*
(*proof*) $\perp_{\mathbf{DTI}}$ is the terminal object of $\mathbf{DTI}$. Easy using Fact 1.

(*end of proof*)

FACT 2. (Theorem 2 of [SmP82]) *Let $\mathbf{K}$ be an $\mathbf{O}$-category and $\Delta = (A_i, g_i)$ be an ω-chain in $\mathbf{K}^E$. Then $\Delta^{Rev} = (A_i, g_i^P)$ is an ω^{op}-chain in $\mathbf{K}$. If Δ^{Rev} has a limit in $\mathbf{K}$, then Δ has a colimit in $\mathbf{K}^E$.*

DEFINITION. (Definition 3 of [SmP82]) A category, $\mathbf{K}$, is an *ω-category* if and only if it has an initial object, and every ω-chain has a colimit.

PROPOSITION 9. *$\mathbf{DTI}^E$ is an ω-category.*
(*proof*) From Proposition 6, Proposition 7, 8 and Fact 2.

(*end of proof*)

DEFINITION. (Definition 4 of [SmP82]) Let $F : \mathbf{K} \to \mathbf{L}$ be a functor. It is *ω-continuous* if and only if it preserves ω-colimits, that is, when Δ is an ω-chain and (A, g) is its limit, then $(F(A), F(g))$ is the limit of $F(\Delta)$.

FACT 3. (Derived from Lemma1, Lemma2 of [SmP82]) *Let $\mathbf{K}$ be a ω-category and let $F : \mathbf{K} \to \mathbf{K}$ be an ω-continuous functor. Let $1_{\mathbf{K}}$ be the initial object of $\mathbf{K}$, and 1_A be the unique morphism from $1_{\mathbf{K}}$ to A. Define the ω-chain Δ to be $(F^n(1_{\mathbf{K}}), F^n(1_{F(1_{\mathbf{K}})}))$ and (A, g) to be the colimit of Δ. Then $F(A) \cong A$.*

DEFINITION. (Definition 10 of [SmP82]) Let $\mathbf{K}$, $\mathbf{L}$, $\mathbf{M}$ be $\mathbf{O}$-categories. A covariant functor $T : \mathbf{K}^{OP} \times \mathbf{L} \to \mathbf{M}$ is *locally continuous* if and only if it is ω-continuous on the hom-sets; that is, if $f_n : A \to B$ is an increasing ω-sequence in $\mathbf{K}^{OP}$ and $g_n : C \to D$ is one in $\mathbf{L}$ then $T(\sqcup_n f_n, \sqcup_n g_n) = \sqcup_n T(f_n, g_n)$.

FACT 4. (Derived from Theorem 3 and Corollary to Theorem 2 of [SmP82]) *Suppose a covariant functor $T : \mathbf{K}^{OP} \times \mathbf{L} \to \mathbf{M}$ is locally continuous, and every ω^{OP}-chain has a limit both in $\mathbf{K}$ and $\mathbf{L}$. Then the functor $T^E : \mathbf{K}^E \times \mathbf{L}^E \to \mathbf{M}^E$ defined as $T^E(A, B) = T(A, B)$ for objects and $T^E(f, g) = T((f^P)^{OP}, g)$ for morphisms is ω-continuous.*

Using Fact 4, We will prove that an functor $F : \mathbf{DTI}^E \to \mathbf{DTI}^E$ defined for objects as $F(M) = M_B + [M \to M]$ is ω-continuous, and that their exists an I-domain M which is isomorphic to $F(M)$ using Fact 3.

PROPOSITION 10. *Let* $M_i = (D_i, T_i, \tau_i)$ $(i = 1,2,3,4)$ *be I-domains and* $f = (f^D, f^T) : M_1 \to M_2$, $g = (g^D, g^T) : M_3 \to M_4$ *be homomorphisms.*
1)Define $\phi^D : [D_2 \to D_3] \to [D_1 \to D_4]$ *and* $\phi^T : [T_2 \to T_3] \to [T_1 \to T_4]$ *as follows:*

$$\phi^D(d : [D_2 \to D_3]) = g^D \cdot d \cdot f^D$$
$$\phi^T(u : [T_2 \to T_3]) = g^T \cdot u \cdot f^T.$$

Then $\phi = (\phi^D, \phi^T)$ *is a homomorphism from* $[M_2 \to M_3]$ *to* $[M_1 \to M_4]$.
2)We can define a functor $HOM : \mathbf{DTI}^{OP} \times \mathbf{DTI} \to \mathbf{DTI}$ *as follows:*

$$HOM(M_1, M_2) = [M_1 \to M_2]$$
$$HOM(f, g) = \phi.$$

(*proof*) 1)Since ϕ^D is continuous, we only have to prove that $\tau_1\tau_4^* \cdot \phi^D = \phi^T \cdot \tau_2\tau_3^*$. Suppose $d \in [D_2 \to D_3]$, $t \in T_1$.

$$\begin{aligned}
(\tau_1\tau_4^* \cdot \phi^D)(d)(t) &= \tau_4(\phi^D(d)(x)) && ;;\textit{Definition of } \tau_1\tau_4^* \\
&\quad \textit{where } x \in D_1 \textit{ such that } \tau_1(x) = t \\
&= \tau_4(g^D \cdot d \cdot f^D(x)) && ;;\textit{Definition of } \phi^D \\
&= (g^T \cdot \tau_3 \cdot d \cdot f^D)(x) && ;;g \textit{ is a homomorphism.} \\
(\phi^T \cdot \tau_2\tau_3^*)(d)(t) &= (g^T \cdot \tau_2\tau_3^*(d) \cdot f^T)(t) && ;;\textit{Definition of } \phi^T \\
&= (g^T \cdot \tau_2\tau_3^*(d) \cdot \tau_2)(f^D(x)) && ;;f \textit{ is a homomorphism.} \\
&\quad \textit{where } x \in D_1 \textit{ such that } \tau_1(x) = t \\
&= (g^T \cdot \tau_3)(d(f^D(x))) && ;;\textit{Definition of } \tau_2\tau_3^*.
\end{aligned}$$

2)Easy.

(*end of proof*)

PROPOSITION 11. *1)The functor* $HOM : \mathbf{DTI}^{OP} \times \mathbf{DTI} \to \mathbf{DTI}$ *defined above is locally continuous.*
2)The product functor $PROD : \mathbf{DTI} \times \mathbf{DTI} \to \mathbf{DTI}$ *is locally continuous.*
3)The sum functor $SUM : \mathbf{DTI}_\perp \times \mathbf{DTI}_\perp \to \mathbf{DTI}_\perp$ *is locally continuous. Where* $\mathbf{DTI}_\perp$ *is the subcategory of* $\mathbf{DTI}$ *whose objects are I-domains and whose morphisms are strict homomorphisms.*

(*proof*) 1) Let $f_n = (f_n^D, f_n^T) : M_1 \to M_2$ be an increasing ω-sequence in $\mathbf{DTI}^{OP}$, and $g_n = (g_n^D, g_n^T) : M_3 \to M_4$ be one in $\mathbf{DTI}$, We only have to prove the equality of D-components, that is, $\sqcup_n (g_n^D \cdot d \cdot f_n^D) = (\sqcup_n g_n^D) \cdot d \cdot (\sqcup_n f_n^D)$, which is obvious.
2),3) Easy.

(*end of proof*)

COROLLARY. *The functors* $HOM^E, PROD^E, SUM^E : \mathbf{DTI}^E \times \mathbf{DTI}^E \to \mathbf{DTI}^E$ *are* ω*-continuous.*

(*proof*) For HOM^E, it is a direct consequence of Proposition 11 and Fact 4. We can consider $PROD$ be a functor from $\mathbf{ID}^{OP} \times (\mathbf{DTI} \times \mathbf{DTI})$ to $\mathbf{DTI}$, and SUM be one from $\mathbf{ID}^{OP} \times (\mathbf{DTI}_\perp \times \mathbf{DTI}_\perp)$ to $\mathbf{DTI}_\perp$, where $\mathbf{ID}$ is the category with one object and one arrow. By Fact 4, $PROD^E$ is a functor from $\mathbf{DTI}^E \times \mathbf{DTI}^E \to \mathbf{DTI}^E$, and SUM^E is a functor from ${\mathbf{DTI}_\perp}^E \times {\mathbf{DTI}_\perp}^E \to {\mathbf{DTI}_\perp}^E$. ${\mathbf{DTI}_\perp}^E$ is isomorphic to $\mathbf{DTI}^E$.

(*end of proof*)

PROPOSITION 12. *The functor* $F : \mathbf{DTI}^E \to \mathbf{DTI}^E$ *defined as*

$$F(M) = M_B + [M \to M]$$
$$F(f : M_1 \to M_2) = SUM^E(id_{M_B}, HOM^E(f, f)).$$

is ω*-continuous.*

THEOREM 3. *1)* M*, the colimit of* $\Delta = (F^n(\perp_{\mathbf{DTI}}), F^n(\perp_{F(\perp)}))$ *in* $\mathbf{DTI}^E$*, and* $M_B + [M \to M]$ *are isomorphic; that is, there are homomorphisms*

$$\begin{array}{lrcl} \Phi = (\Phi^D, \Phi^T) : & M & \to & M_B + [M \to M] \\ \Psi = (\Psi^D, \Psi^T) : & M_B + [M \to M] & \to & M \end{array}$$

which satisfies

$$\Phi \cdot \Psi = id_{(M_B + [M \to M])}$$
$$\Psi \cdot \Phi = id_M.$$

2) Moreover, Φ *and* Ψ *are additive.*

(*proof*) 1) From Proposition 9, 12 and Fact 3.

2) Since Φ and Ψ are morphisms in $\mathbf{DTI}^E$, they are embeddings in **DTI**. In **DTI**, all the embeddings are additive, because it holds in cpo.

(*end of proof*)

4. The syntax of LTI

In this section, we specify the syntax of the language LTI. Before that, we introduce some notions on posets which we use in giving semantics to LTI in the following sections.

4.1 Preliminaries

DEFINITION. Let T be a poset. An element t is *finite* if for every directed subset U of T whose l.u.b. is greater than t, there exists an element $u \in U$ such that $t \leq u$. The set of all finite elements of T is denoted by $\mathcal{F}(T)$.

DEFINITION. Let T be a poset. A subset V of T is *open* if it satisfies the following conditions: (1) if $V \ni t$ then $V \ni u$ for all $u \geq t$. (2) for every directed subset U of T which has the l.u.b. in V, U and V intersects. The set of all open sets of T is denoted by $\mathcal{O}(T)$.

PROPOSITION 13. *Let* T *be a poset.*
1) If $t_1 \sqcup t_2$ *exists for* $t_1, t_2 \in \mathcal{F}(T)$ *then* $t_1 \sqcup t_2 \in \mathcal{F}(T)$.
2) Let $u \in \mathcal{F}(T)$. *Then* $u\Uparrow = \{t \in T | t \succeq u\}$ *is an open set of* T.

DEFINITION. Let u be a finite element of T. Then we call $u\Uparrow$ in Proposition 13 the *principal open set of* T *generated by* u. The set of principal open sets and the empty set is denoted by $\mathcal{P}(T)$.

PROPOSITION 14. *Let* T *be a poset and* T' *be a poset with least element null, and let* $O \in \mathcal{P}(T), u \in \mathcal{F}(T')$. *Then we define* $\lfloor O \mapsto u \rfloor : T \to T'$ *as follows.*

$$\lfloor O \mapsto u \rfloor(t) = \textit{if } (t \in O) \textit{ then } u \textit{ else } null$$

$\lfloor O \mapsto u \rfloor$ *is a finite element of* $\langle T \to T' \rangle$.

PROPOSITION 15. *Let $M = (D, T, \tau)$, $M' = (D', T', \tau')$ be I-domains and let nil and null be the least elements of D' and T' respectively.*
(1) Let $u \in \mathcal{F}(T)$, $v \in T'$, and $f : Val(u) \to Val(v)$. We define $f\triangle^D : D \to D'$ and $f\triangle^T : T \to T'$ as follows.

$$f\triangle^D(x) = \textit{if } (\tau(x) \geq u) \textit{ then } f(x|_u) \textit{ else nil}$$
$$f\triangle^T(t) = \textit{if } (t \geq u) \textit{ then } v \textit{ else null}$$

Then $f\triangle = (f\triangle^D, f\triangle^T)$ is a homomorphism from M to M'.
(2) Let $f = (f^D, f^T)$ be a homomorphism from M to M', and $O \in \mathcal{O}(T)$. We define $f^D!_O : D \to D'$ and $f^T!_O : T \to T'$ as follows,

$$f^D!_O(x) = \textit{if } (\tau(x) \in O) \textit{ then } f^D(x) \textit{ else nil}$$
$$f^T!_O(t) = \textit{if } (t \in O) \textit{ then } f^T(t) \textit{ else null.}$$

Then $f!_O = (f^D!_O, f^T!_O)$ is a homomorphism from M to M'.

Let T be a poset, T' be a poset with least element *null*, $O \in \mathcal{P}(T)$, and f be a function from T to T'. We also write $f!_O$ for the function defined as

$$f!_O(t) = \text{if } (t \in O) \text{ then } f(t) \text{ else } null.$$

Let $M = (D, T, \tau)$ and $M' = (D', T', \tau')$ be I-domains, and let $O \in \mathcal{P}(T)$ and $u \in \mathcal{F}(T')$. Take a $z \in D'$ such that $\tau'(z) = u$ and define $f^D = lambda(x).z$ and $f^T = lambda(t).u$. Then (f^D, f^T) is a homomorphism from M to M'. By Proposition 15, $f!_O$ is a homomorphism from M to M'. You can see that $f^T!_O$ is equal to $\lfloor O \mapsto u \rfloor$. It follows that $\lfloor O \mapsto u \rfloor$ is a member of $[T \to T]$.

4.2 The syntax of LTI

The syntax of LTI consists of Exp: the set of expressions, TExp: the set of type expressions, and typeof: a function from Exp to TExp.

This language depends on the choice of following sets of symbols.

V	:The set of variables
C	:The set of constants
TV	:The set of type variables
TC	:The set of type constants
FTC	:The set of finite type constants

We assume the existence of one to one correspondence typeofv from V to TV. C contains a special constant nil. FTC is a subset of TC and contains a special type constant null. FTC is the set of type constants which are assigned finite types as their meaning. In most practical cases, FTC is same as TC.

First, we define Exp: the set of *expressions*. Exp is the set of untyped lambda-expressions with constants from C.

$$c \in \mathsf{C}$$
$$v \in \mathsf{V}$$
$$e, e_1, e_2 \in \mathsf{Exp}$$
$$e ::= c \mid v \mid \lambda v.\ e \mid e_1\ e_2$$

We use parentheses additionally in order to make the order of association clear.

Next, we define TExp: the set of *type expressions*. We also define FTExp: the set of *finite type expressions* and OExp: the set of *open set expressions* along with defining TExp. In the semantics, a finite type expression denotes a finite element of the type domain, and an open set expression denotes a principal open set of the type domain.

$$
\begin{aligned}
tc &\in \mathsf{TC} \\
tv &\in \mathsf{TV} \\
ftc &\in \mathsf{FTC} \\
te, te_1, te_2 &\in \mathsf{TExp} \\
o, A_i &\in \mathsf{OExp} \quad (i = 0, 1, \ldots, m) \\
fte, B_i &\in \mathsf{FTExp} \quad (i = 0, 1, \ldots, m)
\end{aligned}
$$

$$te ::= tc \mid tv \mid \Lambda tv : o.\ te \mid te_1\ te_2 \mid te_1 \vee te_2$$

$$fte ::= ftc \mid (A_0 \to B_0, A_1 \to B_1, \ldots, A_m \to B_m)$$

$$o ::= fte\uparrow$$

$te_1 \vee te_2$ is a type expression which denotes the least upper bound of the types denoted by te_1 and te_2. $\Lambda tv : o.\ te$ is a type expression which denotes the type of a function whose return value belongs to the type denoted by te when the type of the given argument is a member of the set denoted by o, and otherwise returns *nil*, the value denoted by nil. We abbreviate $\Lambda tv : \mathsf{null}\uparrow\ .\ te$ as $\Lambda tv.\ te$. $(A_0 \to B_0, A_1 \to B_1, \ldots, A_m \to B_m)$ is an abbreviation for $(\Lambda tv : A_1.\ B_1) \vee (\Lambda tv : A_2.\ B_2) \vee \ldots \vee (\Lambda tv : A_m.\ B_m)$. Thus, FTExp is a subset of TExp. Closed expressions and closed type expressions are defined in the usual manner.

Finally, we define the function typeof from Exp to TExp. Suppose that there is a function typeofc from C to the set of closed type expressions such that typeofc(nil) = null. typeof is defined as follows;

$$
\begin{aligned}
\mathsf{typeof}(c) &= \mathsf{typeofc}(c) \quad \text{where } c \in \mathsf{C} \\
\mathsf{typeof}(v) &= \mathsf{typeofv}(v) \quad \text{where } v \in \mathsf{V} \\
\mathsf{typeof}(\lambda v.\ e) &= \Lambda \mathsf{typeofv}(v).\ \mathsf{typeof}(e) \\
\mathsf{typeof}(e_1\ e_2) &= \mathsf{typeof}(e_1)\ \mathsf{typeof}(e_2).
\end{aligned}
$$

We call typeof(e) the type of e.

The syntax of LTI depends on what we take as the symbols and how we define typeofv and typeofc. Here is an example of them.

$$
\begin{aligned}
\mathsf{V} &= \{x\} \\
\mathsf{C} &= \mathsf{INT} + \mathsf{REAL} + \{\mathsf{nil}\} + \{\mathsf{sqrt}, \mathsf{trunc}, +\} \\
\mathsf{TV} &= \{t\} \\
\mathsf{TC} &= \mathsf{FTC} = \{\mathsf{int}, \mathsf{real}, \mathsf{null}\}
\end{aligned}
$$

where INT is the set of symbols denoting integers such as -2, -1, 0, 1, 2, and REAL is the set of symbols denoting (a subset of) real numbers such as 1.0, 2.0, 2.45.

typeofv and typeofc are defined as follows;

$$\begin{aligned}
\mathsf{typeofv}(x) &= t \\
\mathsf{typeofc}(e) &= \mathsf{int} \qquad \text{where } e \in \mathsf{INT} \\
\mathsf{typeofc}(e) &= \mathsf{real} \qquad \text{where } e \in \mathsf{REAL} \\
\mathsf{typeofc}(\mathsf{sqrt}) &= (\mathsf{real}\uparrow \to \mathsf{real}) \\
\mathsf{typeofc}(\mathsf{trunc}) &= (\mathsf{real}\uparrow \to \mathsf{int}) \\
\mathsf{typeofc}(+) &= (\mathsf{int}\uparrow \to (\mathsf{int}\uparrow \to \mathsf{int}), \\
&\qquad \mathsf{real}\uparrow \to (\mathsf{real}\uparrow \to \mathsf{real})).
\end{aligned}$$

The followings are expressions:

$$\begin{aligned}
&((+ \quad 2\ )\ 3) \\
&\lambda x.\ x \\
&\lambda x.\ (x\ x) \\
&\lambda x.\ ((+\ (\mathsf{trunc}\ (\mathsf{sqrt}\ x)))\ x).
\end{aligned}$$

The types of these expressions are as follows;

$$\begin{aligned}
\mathsf{typeof}((+\ 2)\ 3) &= ((+'\ \mathsf{int})\ \mathsf{int}) \\
\mathsf{typeof}(\lambda x.\ x) &= \Lambda t.\ t \\
\mathsf{typeof}(\lambda x.\ (x\ x)) &= \Lambda t.\ (t\ t) \\
\mathsf{typeof}(\lambda x.\ ((+\,(\mathsf{trunc}\,(\mathsf{sqrt}\ x)))\ x)) &= \Lambda t.\ ((+'\ (\mathsf{trunc}'\ (\mathsf{sqrt}'\ x)))\ x).
\end{aligned}$$

Here, we used the following abbreviations:

$$\begin{aligned}
\mathsf{sqrt}' &= (\mathsf{real}\uparrow \to \mathsf{real}) \\
\mathsf{trunc}' &= (\mathsf{real}\uparrow \to \mathsf{int}) \\
+' &= (\mathsf{int}\uparrow \to (\mathsf{int}\uparrow \to \mathsf{int}), \\
&\qquad \mathsf{real}\uparrow \to (\mathsf{real}\uparrow \to \mathsf{real})).
\end{aligned}$$

5. Semantics

In this section, we give a denotational semantics to LTI. The semantic domain is an I-domain $M = (D, T, \tau)$ which satisfies the equation

$$M \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} M_B + [M \to M] + \mathit{WRONG}$$

where $M_B = (D_B, T_B, \tau_B)$ is a given I-domain which consists of the set of fundamental values D_B, the set of fundamental types T_B, and the type assignment function τ_B from D_B to T_B; WRONG is an I-domain whose value domain is $\{\mathit{DWrong}, \mathit{nil}\}$ and type domain is $\{\mathit{TWrong}, \mathit{null}\}$. The semantics is given by a pair of functions $(\mathcal{E}, \mathcal{E}^T)$; $\mathcal{E}$ is a semantic function for expressions, and $\mathcal{E}^T$ is a semantic function for type expressions. We construct them assuming that a semantic function for constants $\mathcal{E}^C : \mathsf{C} \to D$, and a semantic function for type constants $\mathcal{E}^{TC} : \mathsf{TC} \to T$ are given †.

† Note that $\mathcal{E}^{TC}$ is an function to T_B in the examples given in the following sections.

DWrong and *TWrong* mean type mismatch; that is, *DWrong* means that an element of D_B is applied, as a function, to an argument, and *TWrong* means that an element of T_B is applied, as a function, to an argument. There is another notion of 'error' in this semantics; that is, not every type expression has meaning in T. TExp includes a type expression $te_1 \vee te_2$ which denotes the least upper bound of the two types denoted by te_1 and te_2. However, in the type domain T, not every two types have the least upper bound. So $te_1 \vee te_2$ may not have meaning in T. Neither may $\Lambda tv : o.te$ have meaning in T, since one can express functions on types which are not types of generic functions. Hence, $\mathcal{E}^T$ returns *Error*, which is a special element not in T, when the type expression does not have meaning in T. Theorem 4 ensures that the meaning of a type expression is not *Error* if it is a type of an expression. Because we are only interested in types of expressions, we may safely say that T is the semantic domain of type expressions. Note that *Error* is not an element of T though *TWrong* is.

We call $\mathsf{Env} = \mathsf{V} \to D$ the set of environments and $\mathsf{TEnv} = \mathsf{TV} \to T$ the set of type environments. $\mathcal{E}$ and $\mathcal{E}^T$ has the following domains and codomains;

$$\mathcal{E} : \mathsf{Exp} \to (\mathsf{Env} \to D)$$
$$\mathcal{E}^T : \mathsf{TExp} \to (\mathsf{TEnv} \to T + \{Error\}).$$

In the following, we use $[\![,]\!]$ instead of (,) for arguments in syntactic domains. Let *nil* be the bottom of D, *null* be the bottom of T.

First, we define $\mathcal{E}$: the semantic function for expressions. Let $\mathcal{E}^C : \mathsf{C} \to D$ be a given semantic function for constants. $\mathcal{E} : \mathsf{Exp} \to (\mathsf{Env} \to D)$ is defined following the structure of expressions:

$$\mathcal{E}[\![c]\!](\rho) = \mathcal{E}^C[\![c]\!]$$
$$\mathcal{E}[\![v]\!](\rho) = \rho(v)$$
$$\mathcal{E}[\![e_1\ e_2]\!](\rho) = \text{if } (\Phi^D(\mathcal{E}[\![e_1]\!](\rho)) \in [D \to D]) \text{ then } \Phi^D(\mathcal{E}[\![e_1]\!](\rho))(\mathcal{E}[\![e_2]\!](\rho)) \text{ else } DWrong$$
$$\mathcal{E}[\![\lambda v.e]\!](\rho) = \Psi^D(f) \text{ where } f = lambda(d).\mathcal{E}[\![e]\!](\rho[d/v])$$

where $\rho[\![d/v]\!]$ is the environment identical to ρ except that it maps v to d. We must prove that the f appeared in the above definition belongs to $DF = [D \to D]$.

LEMMA. *$lambda(d).\mathcal{E}[\![e]\!](\rho[d/v])$ is a generic function from D to D.*
(*proof*) It is proved using structural induction on the formation of Exp.

(*end of proof*)

Next, we define $\mathcal{E}^T$: the semantic function for type expressions. Let $\mathcal{E}^{TC} : \mathsf{TC} \to T$ be a given semantic function for type constants that maps each element in FTC to $\mathcal{F}(T)$. Before defining $\mathcal{E}^T$, we define $\mathcal{E}^F$: the semantic function for FTExp, and $\mathcal{E}^O$: the semantic function for OExp. Their domains and codomains are as follows;

$$\mathcal{E}^F : \mathsf{FTExp} \to \mathcal{F}(T) + Error$$
$$\mathcal{E}^O : \mathsf{OExp} \to \mathcal{P}(T).$$

$$\mathcal{E}^F[\![ftc]\!] = \mathcal{E}^{TC}[\![ftc]\!]$$
$$\mathcal{E}^F[\![(A_0 \to B_0, A_1 \to B_1, \ldots, A_m \to B_m)]\!]$$

$$= \text{if}(\exists i.\mathcal{E}^F[\![B_i]\!] = Error) \quad \text{then } Error$$
$$\text{else if } (\sqcup_i \lfloor \mathcal{E}^O[\![A_i]\!] \mapsto \mathcal{E}^F[\![B_i]\!] \rfloor \text{ exists})$$
$$\text{then } \Psi^D(\sqcup_i \lfloor \mathcal{E}^O[\![A_i]\!] \mapsto \mathcal{E}^F[\![B_i]\!] \rfloor)$$
$$\text{else } Error$$

$$\mathcal{E}^O[\![fte\uparrow]\!] = \text{if } (\mathcal{E}^F[\![fte]\!] = Error \ ) \text{ then empty set}$$
$$\text{else } \mathcal{E}^F[\![fte]\!]\Uparrow$$

$\mathcal{E}^T : \mathsf{TExp} \to (\mathsf{TEnv} \to T + \{Error\})$ is defined following the structure of expressions:

$$\mathcal{E}^T[\![tc]\!](\rho t) = \mathcal{E}^{TC}[\![tc]\!]$$

$$\mathcal{E}^T[\![tv]\!](\rho t) = \ \rho t(tv)$$

$$\mathcal{E}^T[\![te_1 \ te_2]\!](\rho t) = \text{if } (\mathcal{E}^T[\![te_1]\!](\rho t) = Error \text{ or } \mathcal{E}^T[\![te_2]\!](\rho t) = Error)$$
$$\text{then } Error$$
$$\text{else if } (\Phi^T(\mathcal{E}^T[\![te_1]\!](\rho t)) \in [T \to T])$$
$$\text{then } \Phi^T(\mathcal{E}^T[\![te_1]\!](\rho t))(\mathcal{E}^T[\![te_2]\!](\rho t))$$
$$\text{else } TWrong$$

$$\mathcal{E}^T[\![\Lambda tv : o.te]\!](\rho t) = \text{if } (\mathcal{E}^T[\![te]\!](\rho t[u/tv]) = Error \text{ for } \exists\ u \in T)$$
$$\text{then } Error$$
$$\text{else let } f = lambda(u).\mathcal{E}^T[\![te]\!](\rho t[u/tv])$$
$$\text{if } (f!_{\mathcal{E}^O[\![o]\!]} \in [T \to T])$$
$$\text{then } \Psi^T(f!_{\mathcal{E}^O[\![o]\!]})$$
$$\text{else } Error$$

$$\mathcal{E}^T[\![te_1 \vee te_2]\!](\rho t) = \text{if } (\mathcal{E}^T[\![te_1]\!](\rho t) = Error \text{ or } \mathcal{E}^T[\![te_2]\!](\rho t) = Error \ )$$
$$\text{then } Error$$
$$\text{else if } (\mathcal{E}^T[\![te_1]\!](\rho t) \sqcup \mathcal{E}^T[\![te_2]\!](\rho t) \text{ exists })$$
$$\text{then } \mathcal{E}^T[\![te_1]\!](\rho t) \sqcup \mathcal{E}^T[\![te_2]\!](\rho t)$$
$$\text{else } Error$$

PROPOSITION 16 . *$\mathcal{E}^F$ is the restriction of $\mathcal{E}^T$ to* FTExp *with respect to the embedding functions from* FTExp *to* TExp *and from $\mathcal{F}(T)$ to T.*

$$\begin{array}{ccc} \mathsf{FTExp} & \subset & \mathsf{TExp} \\ \downarrow \mathcal{E}^F & & \downarrow \mathcal{E}^T \\ \mathcal{F}(T) & \subset & T \end{array}$$

(*proof*) It is proved using structural induction on the formation of FTExp.

(*end of proof*)

In this way, we can give denotational semantics to expressions and type expressions on the value domain and on the type domain, respectively. But we are only interested in the case that the meaning of typeof is given by τ. Let ρ be an environment. We define the type environment $\rho^\#$ corresponding to ρ as follows;

$$\rho^\#(tv) = \tau(\rho(v)) \qquad \text{where } \mathsf{typeofv}(v) = tv.$$

THEOREM 4 . *Suppose that every constant c satisfies $\mathcal{E}^T[\![\mathsf{typeofc}(c)]\!] = \tau(\mathcal{E}^C[\![c]\!])$. Then every expression e satisfies $\mathcal{E}^T[\![\mathsf{typeof}(e)]\!](\rho^\#) = \tau(\mathcal{E}[\![e]\!](\rho))$. Especially, $\mathcal{E}^T[\![\mathsf{typeof}(e)]\!](\rho^\#)$ is not Error.*

(*proof*) It is proved using structural induction on the formation of Exp.

(*end of proof*)

This theorem shows that when $\mathcal{E}^C$ and $\mathcal{E}^{TC}$ satisfies $\mathcal{E}^T \cdot \mathsf{typeofc} = \tau \cdot \mathcal{E}^C$. The following diagram commutes.

$$\begin{array}{ccccc} \mathsf{Exp} & \times & \mathsf{Env} & \xrightarrow{\mathcal{E}} & D \\ \downarrow \mathsf{typeof} & & \downarrow \# & & \downarrow \tau \\ \mathsf{TExp} & \times & \mathsf{TEnv} & \xrightarrow{\mathcal{E}^T} & T \end{array}$$

Especially for closed expressions, we have

$$\begin{array}{ccc} \mathsf{Exp} & \xrightarrow{\mathcal{E}} & D \\ \downarrow \mathsf{typeof} & & \downarrow \tau \\ \mathsf{TExp} & \xrightarrow{\mathcal{E}^T} & T. \end{array}$$

6. Examples

In this section, we calculate the meaning of expressions and corresponding type expressions shown in section 4.

As in section 4, we fix the symbols of LTI as follows,

$$\begin{aligned} \mathsf{V} &= \{x\} \\ \mathsf{C} &= \mathsf{INT} + \mathsf{REAL} + \{\mathsf{nil}\} + \{\mathsf{sqrt}, \mathsf{trunc}, +\} \\ \mathsf{TV} &= \{t\} \\ \mathsf{TC} &= \mathsf{FTC} = \{\mathsf{int}, \mathsf{real}, \mathsf{null}\} \end{aligned}$$

and the types of variables and constants as follows,

$$\begin{aligned} \mathsf{typeofv}(x) &= t \\ \mathsf{typeofc}(e) &= \mathsf{int} \qquad \text{where } e \in \mathsf{INT} \\ \mathsf{typeofc}(e) &= \mathsf{real} \qquad \text{where } e \in \mathsf{REAL} \\ \mathsf{typeofc}(\mathsf{sqrt}) &= (\mathsf{real}\uparrow \to \mathsf{real}) \\ \mathsf{typeofc}(\mathsf{trunc}) &= (\mathsf{real}\uparrow \to \mathsf{int}) \\ \mathsf{typeofc}(+) &= (\mathsf{int}\uparrow \to (\mathsf{int}\uparrow \to \mathsf{int}), \\ &\qquad \mathsf{real}\uparrow \to (\mathsf{real}\uparrow \to \mathsf{real})). \end{aligned}$$

We define the I-domain $M_B = (D_B, T_B, \tau_B)$ as follows,

$$\begin{aligned} |T_B| &= \{int, real, null\} \\ &\qquad int \geq real, real \geq null \\ |D_B| &= INT + REAL + \{nil\} \\ &\qquad 1 \succeq 1.0, 2 \succeq 2.0, \ldots, \\ &\qquad x \succeq nil \qquad \text{where } x \in REAL \end{aligned}$$

$$\begin{aligned} \tau_B(x) &= int \qquad \text{where } x \in INT \\ \tau_B(x) &= real \qquad \text{where } x \in REAL \\ \tau_B(nil) &= null, \end{aligned}$$

where INT is the set of integers and $REAL$ is the set of real numbers. It is easy to verify that M_B is an I-domain.

From $M_B = (D_B, T_B, \tau_B)$, we can construct an I-domain $M = (D, T, \tau)$ which satisfies

$$M \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} M_B + [M \to M] + WRONG.$$

In this example, we identify elements of D_B with elements of D through Φ^D and Ψ^D. We also identify elements of T_B with elements of T through Φ^T and Ψ^T.

Let $\mathcal{E}^C : \mathsf{C} \to D$ be a function defined as follows,

$\mathcal{E}^C$ is defined obviously to elements of INT and REAL.
$\mathcal{E}^C[\![\mathsf{nil}]\!] = nil$
$\mathcal{E}^C[\![\mathsf{sqrt}]\!] = \Psi^D(sqrt\triangle)$
where $sqrt : REAL \to REAL$ is a function which calculates the square root.
$\mathcal{E}^C[\![\mathsf{trunc}]\!] = \Psi^D(trunc\triangle)$
where $trunc : REAL \to INT$, is a function which calculates the truncation.

$$\mathcal{E}^C[\![+]\!] = \Psi^D(f \sqcup g)$$
$$\text{where } f = lambda(x).\ \text{if } (\tau(x) = int) \text{ then } \Psi^D(h) \text{ else } nil$$
$$\text{where } h = lambda(y).\ \text{if } (\tau(y) = int) \text{ then } x +_i y \text{ else } nil$$
$$g = lambda(x).\ \text{if } (\tau(x) \geq real) \text{ then } \Psi^D(h) \text{ else } nil$$
$$\text{where } h = lambda(y).\ \text{if } (\tau(y) \geq real) \text{ then } x|_{real} +_r y|_{real} \text{ else } nil$$

where $+_r : REAL \times REAL \to REAL$
$+_i : INT \times INT \to INT$
are addition functions on $REAL$ and INT respectively.

Let $\mathcal{E}^{TC} : TC \to T$ be a function defined as follows,

$$\mathcal{E}^{TC}[\![\mathsf{int}]\!] = int$$
$$\mathcal{E}^{TC}[\![\mathsf{real}]\!] = real$$
$$\mathcal{E}^{TC}[\![\mathsf{null}]\!] = null.$$

$\mathcal{E}$ and $\mathcal{E}^T$ are defined from $\mathcal{E}^C$ and $\mathcal{E}^{TC}$ as in the previous section. It is easy to verify that every constant c satisfies $\mathcal{E}^T[\![\mathsf{typeofc}(c)]\!] = \tau(\mathcal{E}^C[\![c]\!])$. In the following examples, we omit environment arguments for closed expressions and closed type expressions.

EXAMPLE 1. ((+ 2) 3)

$$\mathsf{typeof}((+\ 2)\ 3) = ((+'\ \mathsf{int})\ \mathsf{int})$$
$$\text{where } +' = (\mathsf{int}\uparrow \to (\mathsf{int}\uparrow \to \mathsf{int})$$
$$\mathsf{real}\uparrow \to (\mathsf{real}\uparrow \to \mathsf{real})).$$

$$\mathcal{E}[\![(+\ 2)]\!] = \Phi^D(\mathcal{E}[\![+]\!])(\mathcal{E}[\![2]\!])$$
$$= (f \sqcup g)(2) \text{ where } f \text{ and } g \text{ are as in the definition of } \mathcal{E}[\![+]\!].$$
$$= f(2) \sqcup g(2)$$
$$= \Psi^D(h1) \sqcup \Psi^D(h2) \text{ where}$$

$$h1 = lambda(y).\ \text{if } (\tau(y) = int) \text{ then } 2 +_i y \text{ else } nil$$
$$h2 = lambda(y).\ \text{if } (\tau(y) \geq real) \text{ then } 2.0 +_r y|_{real} \text{ else } nil$$

$$\begin{aligned}
\mathcal{E}[\![((+\ 2)3)]\!] &= \Phi^D(\mathcal{E}[\![(+\ 2)]\!])(\mathcal{E}[\![3]\!]) \\
&= \Phi^D(\Psi^D(h1) \sqcup \Psi^D(h2))(3) \\
&= (\Phi^D\Psi^D(h1) \sqcup \Phi^D\Psi^D(h2))(3) \qquad \text{;strictness of } \Phi^D \\
&= h1(3) \sqcup h2(3) \\
&= 5 \sqcup 5.0 \\
&= 5
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}^T[\![+']\!] = \quad & \Psi^T(\lfloor int\Uparrow \mapsto \Psi^T(\lfloor int\Uparrow \mapsto int\rfloor\rfloor)) \\
\sqcup\ & \Psi^T(\lfloor real\Uparrow \mapsto \Psi^T(\lfloor real\Uparrow \mapsto real\rfloor\rfloor))
\end{aligned}$$

From here on, we identify the elements of $[D \to D]$ with elements of D and the elements of $[T \to T]$ with elements of T for simplicity.

$$= \lfloor int\Uparrow \mapsto \lfloor int\Uparrow \mapsto int\rfloor\rfloor \sqcup \lfloor real\Uparrow \mapsto \lfloor real\Uparrow \mapsto real\rfloor\rfloor$$

$$\begin{aligned}
\mathcal{E}^T[\![(+' \ \mathsf{int})]\!] &= (\ \lfloor int\Uparrow \mapsto \lfloor int\Uparrow \mapsto int\rfloor\rfloor \\
&\qquad \sqcup \lfloor real\Uparrow \mapsto \lfloor real\Uparrow \mapsto real\rfloor\rfloor)(int) \\
&= \lfloor int\Uparrow \mapsto int\rfloor \sqcup \lfloor real\Uparrow \mapsto real\rfloor \\
&\qquad \text{;} int \geq int \text{ and } int \geq real
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}^T[\![((+' \ \mathsf{int})\ \mathsf{int})]\!] &= (\lfloor int\Uparrow \mapsto int\rfloor \sqcup \lfloor real\Uparrow \mapsto real\rfloor)(int) \\
&= int \sqcup real \\
&= int
\end{aligned}$$

EXAMPLE 2. $\lambda x.x$

$$\mathsf{typeof}(\lambda x.x) = \Lambda t.t\ .$$

$$\mathcal{E}[\![x]\!](\rho) = \rho(x)$$

$$\begin{aligned}
\mathcal{E}[\![(\lambda x.x)]\!] &= lambda(d).\mathcal{E}[\![x]\!](\rho[x/d]) \\
&= lambda(d).d
\end{aligned}$$

$$\mathcal{E}^T[\![t]\!](\rho t) = \rho t(t)$$

$$\begin{aligned}
\mathcal{E}^T[\![(\Lambda t.t)]\!] = & \text{if } (\mathcal{E}^T[\![t]\!](\rho t[t/u]) = Error \text{ for } \exists u) \text{ then } Error \\
& \text{else let } f = (lambda(u).\mathcal{E}^T[\![te]\!](\rho t[t/u])) \\
& \quad \text{if } ((f!_{\mathcal{E}^o[\![\mathsf{null}\uparrow]\!]}) \in [T \to T]) \text{ then } f!_{\mathcal{E}^o[\![\mathsf{null}\uparrow]\!]} \\
& \quad \text{else } Error \\
= & \ f!_{null\Uparrow} \text{ where } f = lambda(u).u \\
= & \ lambda(u).u
\end{aligned}$$

EXAMPLE 3. $\lambda x.(x\ x)$

$$\mathsf{typeof}(\lambda x.(x\ x)) = \Lambda t.(t\ t)$$

As example 2, we can calculate

$$\mathcal{E}[\![(\lambda x.(x\ x))]\!] = lambda(d).\text{if } (d \in [D \to D]) \text{ then } d(d) \text{ else } DWrong$$

$$\mathcal{E}^T[\![(\Lambda t.(t\ t))]\!] = lambda(u).\text{if } (u \in [T \to T]) \text{ then } u(u) \text{ else } TWrong$$

EXAMPLE 4. $\lambda x.((+(\mathsf{trunc}(\mathsf{sqrt}\ x)))\ x)$

$$\mathsf{typeof}(\lambda x.((+(\mathsf{trunc}(\mathsf{sqrt}\ x)))\ x)) = \Lambda t.((+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))\ t)$$

where $\mathsf{sqrt}' = (\mathsf{real}\uparrow \to \mathsf{real})$
$\mathsf{trunc}' = (\mathsf{real}\uparrow \to \mathsf{int})$
$+' = (\mathsf{int}\uparrow \to (\mathsf{int}\uparrow \to \mathsf{int}),$
$\mathsf{real}\uparrow \to (\mathsf{real}\uparrow \to \mathsf{real})).$

$$\mathcal{E}[\![(\mathsf{trunc}(\mathsf{sqrt}\ x))]\!](\rho) = trunc\triangle(sqrt\triangle(\rho(x)))$$

Using the fact that
if $(\tau(\rho(x)) \geq real)$ then $\tau(trunc\triangle(sqrt\triangle(\rho(x)))) = int$
else $\tau(trunc\triangle(sqrt\triangle(\rho(x)))) = null,$

$$\begin{aligned}
&\mathcal{E}[\![(+(\mathsf{trunc}(\mathsf{sqrt}\ x)))]\!](\rho)\\
&\quad = \text{if } (\tau(\rho(x)) \geq real)\\
&\qquad \text{then } \quad lambda(y).\ \text{if } (\tau(y) = int)\\
&\qquad\qquad\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x))) +_i y\\
&\qquad\qquad\qquad\qquad \text{else } nil\\
&\qquad\qquad \sqcup\ lambda(y).\ \text{if } (\tau(y) \geq real)\\
&\qquad\qquad\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x)))|_{real} +_r y|_{real}\\
&\qquad\qquad\qquad\qquad \text{else } nil\\
&\qquad \text{else } nil\\
&\quad = \text{if } (\tau(\rho(x)) \geq real)\\
&\qquad \text{then } \quad lambda(y).\ \text{if } (\tau(y) = int)\\
&\qquad\qquad\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x))) +_i y\\
&\qquad\qquad\qquad\qquad \text{else if } (\tau(y) = real)\\
&\qquad\qquad\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x)))|_{real} +_r y\\
&\qquad\qquad\qquad\qquad \text{else } nil\\
&\qquad \text{else } nil
\end{aligned}$$

$$\begin{aligned}
&\mathcal{E}[\![((+(\mathsf{trunc}(\mathsf{sqrt}\ x)))x)]\!](\rho)\\
&\quad = \text{if } (\tau(\rho(x)) \geq real)\\
&\qquad \text{then if } (\tau(\rho(x)) = int)\\
&\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x))) +_i \rho(x)\\
&\qquad\qquad \text{else if } (\tau(\rho(x)) = real)\\
&\qquad\qquad \text{then } trunc\triangle(sqrt\triangle(\rho(x)))|_{real} +_r \rho(x)\\
&\qquad\qquad \text{else } nil\\
&\qquad \text{else } nil
\end{aligned}$$

$$\begin{aligned}
&\mathcal{E}[\![(\lambda x.((+(\mathsf{trunc}(\mathsf{sqrt}\ x)))x))]\!] = lambda(d).\ \text{if } (\tau(d) = int)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{then } trunc(sqrt(d|_{real})) +_i d\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{else if } (\tau(d) = real)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{then } trunc(sqrt(d))|_{real} +_r d\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{else } nil\ .
\end{aligned}$$

$$\mathcal{E}^T[\![\mathsf{sqrt}']\!] = \lfloor real\Uparrow \mapsto real \rfloor$$
$$\mathcal{E}^T[\![\mathsf{trunc}']\!] = \lfloor real\Uparrow \mapsto int \rfloor$$
$$\begin{aligned}
\mathcal{E}^T[\![+']\!] = \quad & \lfloor int\Uparrow \mapsto \lfloor int\Uparrow \mapsto int \rfloor \rfloor\\
& \sqcup \lfloor real\Uparrow \mapsto \lfloor real\Uparrow \mapsto real \rfloor \rfloor
\end{aligned}$$
$$\mathcal{E}^T[\![\ t]\!](\rho t) = \rho t(t)$$
$$\mathcal{E}^T[\![(\mathsf{sqrt}'\ t)]\!](\rho t) = \text{if } (\rho t(t) \geq real) \text{ then } real \text{ else } null$$

$$\mathcal{E}^T[\![(\mathsf{trunc}'(\mathsf{sqrt}'\ t))]\!](\rho t) = \text{if } (\rho t(t) \geq real) \text{ then } int \text{ else } null$$

$$\mathcal{E}^T[\![(+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))]\!](\rho t) = \text{if } (\rho t(t) \geq real) \text{ then } \lfloor int\Uparrow \mapsto int \rfloor \sqcup \lfloor real\Uparrow \mapsto real \rfloor \text{ else } null$$

$$\mathcal{E}^T[\![((+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))t)]\!](\rho t) = \text{if } (\rho t(t) \geq real) \text{ then if } (\rho t(t) \geq int) \text{ then } int \sqcup real \text{ else if } (\rho t(t) \geq real) \text{ then } real \text{ else } null \text{ else } null$$

$$\mathcal{E}^T[\![(\Lambda t.((+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))t))]\!] = lambda(t).\text{if } (t = int) \text{ then } int \text{ else if } (t = real) \text{then } real \text{ else } null.$$

This formula is equal to

$$\lfloor int\Uparrow \mapsto int \rfloor \sqcup \lfloor real\Uparrow \mapsto real \rfloor.$$

So we get the equality:

$$\mathcal{E}^T[\![\Lambda\ t.((+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))t)]\!] = \mathcal{E}^T[\![(\mathsf{int}\uparrow \to \mathsf{int}) \vee (\mathsf{real}\uparrow \to \mathsf{real})]\!].$$

If we can make a reduction system on type expressions which reduces the type expression $(\Lambda t.((+'(\mathsf{trunc}'(\mathsf{sqrt}'\ t)))t))$ to $(\mathsf{int}\uparrow \to \mathsf{int}) \vee (\mathsf{real}\uparrow \to \mathsf{real})$, it will be applied to the type checking of generic functions.

7. Further works

• *Appropriateness of the model* In this model, we defined that the value domain of an I-domain is a cpo and that generic functions are continuous, which we used in solving domain equations. Though these conditions may not be natural, it seems that most type structure and generic functions we are interested in satisfy them. We must continue investigating in the appropriateness of the model.

• *Formal system* We defined only the syntax and the semantics of LTI. In order to extend it to a programming language, we need to define reduction on Exp and TExp. As the expressions of LTI are same as those of lambda-calculus, we can define β-reduction on Exp and it can be proved that β-reduction preserves the meaning of expressions. The author is also interested in reduction on type expressions, because it can be applied to static type checking of generic functions. It is left as an further work.

• *Method combination.* Originally, this research aimed at designing semantically clear way of doing *method combination*; that is, combining generic functions into one generic function by taking the least upper bound. However, it has some difficulties concerning "Errors". This problem will be further investigated in the near future.

• *Extending the type system.* We only defined function types in LTI. It is easy to add product types to LTI and give semantics to it. The author is interested in adding types which are similar to classes in object oriented languages, which will make the type system more powerful and show the effectiveness of generic functions and method combination.

• *The relation with type theory.* In Martin-Lof type theory, they deal with dependent types such as Σ and Π. Π is the type of functions the type of whose return values vary with the value of the argument, in contrast to generic functions, the type of whose return values vary with the type of the argument. The author is wondering whether there is an interesting relation between these concepts. [Rey85] has pointed out some open problems which appear when considering subtypes, such as let construction, type checking algorithm, infinite, recursively defined types. The author believes that formal semantics like the one given here can be the foundation for solving these problems.

Acknowledgement

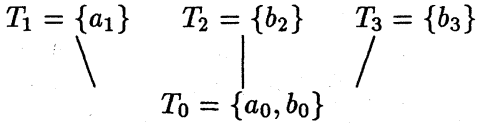
The author would like to express his deep gratitude to Professor Reiji Nakajima for his appropriate advices, and to Mr. Masami Hagiya for many invaluable advices throughout this research and careful reading of the paper.

Appendix

• *An example that τ^* is not a surjective function*
Define that

$$
\begin{aligned}
&M = (T, D, \tau),\\
&T = \{T_0, T_1, T_2, T_3, null\}, T_1 \geq T_0, T_2 \geq T_0, T_3 \geq T_0,\\
&D = \{a_0, b_0, a_1, b_2, b_3, nill\}, a_1 \succeq a_0, b_2 \succeq b_0, b_3 \succeq b_0,\\
&\tau(a_0) = \tau(b_0) = T_0,\\
&\tau(a_1) = T_1,\\
&\tau(b_2) = T_2,\\
&\tau(b_3) = T_3.
\end{aligned}
$$

M is an I-domain.



Define a monotone function $u : T \to T$ that

$$
\begin{aligned}
&u(T_0) = T_0,\\
&u(T_1) = T_1,\\
&u(T_2) = T_1,\\
&u(T_3) = T_3.
\end{aligned}
$$

u is not in the image of τ^*.

• *An example that $[T \to T]$ is not a cpo though T and T' are.*
Define that

$$
\begin{aligned}
&T = \{U, S, T_1, T_2, \ldots, T_w\}\\
&\quad U \leq S, \quad U \leq T_1 \leq \ldots \leq T_n \leq \ldots \leq T_w\\
&Val(U) = \{u_1, u_2, \ldots, u_n, \ldots, u_w\}\\
&Val(S) = \{s_1, s_2, \ldots, s_n, \ldots\}\\
&Val(T_1) = \{t_{1,1}, t_{1,2}, \ldots, t_{1,n}, \ldots, t_{1,w}\}\\
&Val(T_2) = \{\quad t_{2,2}, \ldots, t_{2,n}, \ldots, t_{2,w}\}\\
&\quad \ldots
\end{aligned}
$$

$$Val(T_n) = \{ \qquad t_{n,n},\ldots,t_{n,w}\}$$
$$\ldots$$
$$Val(T_w) = \{ \qquad t_{w,w}\}$$
$$D = Val(U) + Val(S) + Val(T_1) + \ldots + Val(T_w)$$

and define the order on D as

$$s_i \succeq u_i \ (i = 1,2,\ldots,n,\ldots)$$
$$t_{1,i} \succeq u_i \ (i = 1,2,\ldots,n,\ldots,w)$$
$$t_{n,i+1} \succeq t_{n,i} (i = 1,2,\ldots,n,\ldots,w)$$

(D,T,τ) is an I-domain.

$$\begin{array}{lllllllll}
\{ & & & & & t_{w,w} & \} = T_w & & \\
 & & & & & | & | & & \\
 & & & & & \ldots & & & \\
 & & & & & \ldots & & & \\
 & & & & & \ldots & & & \\
 & & & & | & | & | & & \\
\{ & & & t_{n,n}, & t_{n,n+1}, & \ldots, & t_{n,w} & \} = T_n & \\
 & & & | & | & & | & | & \\
 & & & & & & \ldots & & \\
 & & | & | & | & & | & | & \\
\{ & & t_{2,2}, & \ldots, & t_{2,n}, & t_{2,n+1}, & \ldots, & t_{2,w} & \} = T_2 \qquad S = \{s_1,\ldots,s_n,\ldots\} \\
 & & | & & | & | & & | & | \\
\{ & t_{1,1}, & t_{1,2}, & \ldots, & t_{1,n}, & t_{1,n+1}, & \ldots, & t_{1,w} & \} = T_1 \qquad / \\
 & | & | & & | & | & & | & | \\
\{ & u_1, & u_2, & \ldots, & u_n, & u_{n+1}, & \ldots, & u_w & \} = U
\end{array}$$

We show an example of $f_i : T \to T$ $(i = 0,1,2,\ldots)$ in which $\{f_i\}$ is a directed set and $f_i \in [T \to T]$, but the l.u.b. of $F = \{f_i\}$ does not exist in $[T \to T]$. Define f_n $(n = 0,1,2,\ldots)$ as

$$f_n(S) = S$$
$$f_n(U) = U$$
$$f_n(T_i) = T_n \ (i = 1,2,\ldots,n-1)$$
$$f_n(T_i) = T_i \ (i = n,n+1,\ldots,w)$$

f_n are monotonic functions and $\{f_n\}$ is a directed set. Define $g_n : D \to D (n = 0,1,2,\ldots)$ as

$$g_n(s_i) = s_n \quad (i = 1,2,\ldots,n-1)$$
$$g_n(s_i) = s_i \quad (i = n,n+1,\ldots)$$
$$g_n(u_i) = u_n \quad (i = 1,2,\ldots,n-1)$$
$$g_n(u_i) = u_i \quad (i = n,n+1,\ldots,w)$$
$$g_n(t_{k,i}) = t_{n,n} \ (i = 1,2,\ldots,n-1 \quad k = 1,2,\ldots,n-1)$$
$$g_n(t_{k,i}) = t_{n,i} \ (i = n,n+1,\ldots,w \quad k = 1,2,\ldots,n-1)$$
$$g_n(t_{k,i}) = t_{k,i} \ (k = n,n+1,\ldots,w \quad i = n,n+1,\ldots,w)$$

g_n is an generic function and $\tau^*(g_n) = f_n$. $\sqcup F$ satisfies

$$\sqcup F(S) = S$$
$$\sqcup F(U) = U$$
$$\sqcup F(T_i) = T_w \ (i = 1, 2, \ldots, n, \ldots, w)$$

There is no generic function g which satisfies $\tau^*(g) = \sqcup F$.

References

[AiN86] H. Ait-Kaci and R. Nasr (1986), *LOGIN: A logic programming language with built-in inheritance*, in "J. Logic Programming, 3,(3),185-215"

[Ait86] H. Ait-Kaci (1986),*An algebraic semantics approach to the effective resolution of type equations*, in "Theoretical Computer Science, 45, 293-351"

[Bar81] H. P. Barendregt (1981), "The Lambda Calculus: Its Syntax and Semantics", North-Holland, Amsterdam.

[BKK86] D. G. Bobrow, K. Kahn and G. Kiczales (1986),*Commonloops: Merging Common Lisp and Object-Oriented Programming*, in "OOPSLA 86"

[BDG87] D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. Keene, G. Kiczales and D. A. Moon (Feb. 1987) "Common Lisp Object System Specification", ANSI X3 Draft

[BrC87] J. P. Briot and P. Cointe, *A Uniform Model for Object-Oriented Languages Using the Class Abstraction*, "IJCAI'87"

[Car85] L. Cardelli (1985),*A Semantics of Multiple Inheritance*, in "Lecture Notes in Computer Sciences, vol. 173, 51-68"

[CaW85] L. Cardelli and P. Wegner (1985),*On understanding Types, Data Abstraction, and Polymorphism*, in "ACM Computing Survey, vol. 17, No. 4"

[MPS 86] D. B. MacQueen, G. D. Plotokin and R. Sethi (1986),*An Ideal Model for Recursive Polymorphic Types*, in "Information and Control,71,95-130"

[Mey82] A. R. Meyer (1982), *What is a Model of the Lambda Calculus?*, in "Information and Control 32,87-122."

[Mil78] R. Milner (1978),*A Theory of Type Polymorphism in Programming*, in "J. Comput. System. Sci.,17,N0.3, 348-375."

[Rey85] J. C. Reynolds (1985),*Three approaches to type structures*, in "Lecture Notes in Computer Science, vol. 185, 97-138"

[Rey81] J. C. Reynolds (1981),*Using Category Theory to Design Implicit Conversions and Generic Operators*, in "Lecture Notes in Computer Science vol. 94,211-258"

[SmP82] M. B. Smyth and G. D. Plotokin (1982),*The category-theoretic solution of recursive domain equations*, in "SIAM J. Comput.,11,No.4, 761-783."